

RDF and Contexts: Use of SPARQL and Named Graphs to Achieve Contextualization

Heiko Stoermer¹, Ignazio Palmisano², Domenico Redavid², Luigi Iannone², Paolo Bouquet¹, and Giovanni Semeraro²

¹ University of Trento,
Dept. of Information and Communication Tech.,
Trento, Italy

{stoermer, bouquet}@dit.unitn.it

² Università degli Studi di Bari
Dipartimento di Informatica
Bari, Italy

{palmisano, redavid, iannone, semeraro}@di.uniba.it

Abstract. The very simple structure of the RDF data model and semantics can lead to a number of issues when more complex scenarios are supposed to be represented in RDF. Aspects such as temporary evolution of a knowledge base, relevance and trust require more than just a model that consists of a set of universally true statements, without any reference to a situation, a point in time, or generally a *context*. Our proposed solution is to use the notion of *context* to separate statements that refer to different contextual information, which could so far not explicitly be tied to the statements because of the simplicity of RDF. In this paper we describe a practical solution to this problem, which has been implemented in the VIKEF project.

1 Problem Description and Motivation

The VIKEF project deals with creating large-scale information systems that base on Semantic Web technology. At the center of the envisioned systems there is an RDF knowledge base (KB) that contains a large amount of information about documents and their contents. This information is gathered by information and knowledge extraction processes at the base level, semantically enriched and related to ontological knowledge, and then stored in an RDF triple store called RDFCore, which will be described in more detail in Sect. 3.1. On top of this KB, semantic-enabled services will be implemented to provide a next-generation information system.

Current RDF triple stores are built to represent a single bag of RDF triples, i.e. all statements are stored in the same information space together. However, from a Knowledge Representation point of view, RDF statements in general are context-free, and thus follow a notion of *universal truth*, while in our opinion knowledge in an information system is context-dependent. In effect, without a context-based system, it is possible (and probable) that semantically contradictory statements will be stored in the KB, such as for instance “Silvio Berlusconi is the Prime minister of Italy” and “Romano Prodi is the Prime minister of Italy” as the result of knowledge extracted from articles written in different years. These contradictions are however unwanted in a logical system because they would interfere with the higher level reasoning that is to be performed to provide Semantic Web functionality, such as semantic browsing, search, visualization, etc. Additionally, we would like to be able to model other aspects such as relevance, credibility and validity of a statement, all of which require further qualification.

If we think about the Semantic Web as a whole, with a large number of uncoordinated information systems, the problem becomes even more evident. If every peer builds up a KB of unqualified RDF statements, the set of universally true facts in the Semantic Web becomes enormously large and impossible to handle from a semantic point of view. In our opinion, such contradictions, contradictory beliefs and facts that become semantically incorrect in the absence of additional pragmatic or contextual information are likely to impose serious problems on the coordination and interoperation of information systems in the Semantic Web.

2 Context in RDF Knowledge Bases

We think that the mentioned issues can be approached by introducing the notion of *context* into RDF, to limit the scope of an RDF statement to the context in which it is relevant or valid, because in our opinion this is required for anything sensible to be expressed in the Semantic Web. We want to present a mechanism to qualify statements and thus to model that a statement is true *only under a certain set of conditions*, which will help us store information in

the KB that would cause contradictions or inconsistencies in a plain RDF A-Box.

2.1 Context in KR - Multi Context Systems

The theoretical ideas presented in this paper base on the logical theory of *Multi Context Systems* and the principles of *Locality* and *Compatibility* presented e.g. in [6], with influences from [3, 4]. Basically, this theory states that contexts can be seen in a peer-to-peer view, resembling more general aspects such as human beliefs, agent knowledge or distributed systems. Relations between contexts however, i.e. to reason across contexts, are to be expressed in so-called *compatibility relations* (CRs), that formalize how under certain circumstances knowledge from other contexts becomes relevant.

The basic idea is to have all statements that belong to a context in a separate named RDF graph, and extend the RDF semantics in a way to enable contexts to appear as standard objects in RDF statements of other contexts. Then, we want to model the mentioned CRs between contexts, to allow for reasoning across contexts. This aspect is probably the most important one, because from an application perspective it is crucial that sensible queries can be issued and *all* relevant information is taken into account - which requires reasoning across contexts and reasoning on the relations between contexts (i.e. on statements of the form $\langle c_x \text{ R } c_y \rangle$ where c_x and c_y are RDF Contexts, or $\langle f \text{ R } c' \rangle$ respectively $\langle c' \text{ R } f \rangle$ with $f \in c$). We are only starting to explore in full depth the aspects of compatibility relations that are relevant for the VIKEF project.

In order to model CRs, many approaches, such as integrating external ontologies about CRs or providing a default ontology in the architecture, are applicable. However, in our opinion the basic problem with these approaches is the fact that many interesting relations between architectures cannot be fully formalized with the help of a Semantic Web ontology, which is based on Description Logics. As an example, take a relation such as $\langle c' \text{ EXTENDS } c \rangle$, which expresses that context c' represents an extension to context c . The underlying assumption of the *EXTENDS* relation is that the two contexts are compatible, i.e. they agree on the relevant context parameters. The semantics of this relation have to be expressed algorithmically:

if c and c' are compatible, if no answer to a query can be given in c propagate query to c'

But how are these compatibility relations supposed to be modeled? Three of the approaches we foresee to do this have been presented in [1], together with a more detailed analysis and a discussion of related work.

It is our opinion that the approach to be chosen is to implement a CR as a *semantic attachment* [9], which can be thought of as a sort of plugin to the system, one attachment per CR. This has the positive effects that i) there is no restriction on how many and which kind of CRs are part of such a system and ii) implementation of the CRs is generally not restricted to any specific language or system.

2.2 Related Work

The W3C Named Graph Interest Group³ is working on an approach to define a way to represent a graph as an object in an RDF KB. A substantial article has been published in 2005 [2], and implementational results are now part of the Named Graphs API for Jena (NG4J)⁴. This approach has mainly been driven by the need for developing a trust model in RDF, but it could also serve as an underlying implementation in order to provide a base for the CRs discussed above.

3 The Proposed Solution: System Architecture

Our practical solution to context issues is based on the following requirements:

- Easy and simple identification of contexts
- Separate and independent storage for each context
- Easy querying of one or more contexts
- Easy reasoning on context parameters values
- Ability to plug new CRs in the architecture
- Ability to use CRs of higher expressive level, i.e. higher than OWL and/or DL

³ <http://www.w3.org/2004/03/trix/>

⁴ <http://www.wiwiss.fu-berlin.de/suhl/bizer/ng4j/>

As sketched in Fig. 2, the two main parts of our implementation are what is “inside” RDFCore (i.e. the RDF storage level) and “outside” of it. In Sect. 3.1 and Sect. 3.2 we will discuss the details of the architecture.

3.1 RDF Storage

As RDF storage, we use RDFCore: presented in [5], it is a component used for storage and retrieval of RDF graphs, including multi user support and extensible support for query languages.

RDFCore has been adopted in the VIKEF Project as the basic component for RDF metadata storage, where its SOAP⁵-exposed services have been wrapped as a Web Service⁶ for metadata storage, retrieval and querying.

RDFCore also has extensible support for different solutions for physical persistence. At the time of writing, there are three working implementations of *RDFEngineInterface* (the basic interface to be implemented by plugins), two based on the already mentioned Jena Toolkit, one with MySQL RDBMS⁷ as persistent storage, called *RDFEngineJENA*, and the other one using Microsoft SQL Server⁸, called *RDFEngineMsSQL*. The third implementation is based on simple RDF/XML files, and is called *RDFEnginePlain*. All these implementations are based on the Jena API.

More in detail, the main idea behind the *RDFEngineInterface* is to give to applications that use RDFCore a very simple way to store, retrieve and query RDF models. The added value over the existing Jena API, which is in fact richer than *RDFEngineInterface*, is that in *RDFEngineInterface* each method checks that the current application, or better the user that the current application is acting on behalf of, has the right to start the required operations. We see this as a very useful feature in systems in which a remote module is responsible for keeping the data, since it is possible that more than one application have access to the same data, either by mistake or on purpose.

Basic operations in *RDFEngineInterface* are:

⁵ <http://www.w3.org/2000/xp/Group/>

⁶ <http://www.w3.org/2002/ws/>

⁷ <http://dev.mysql.com/doc/mysql/en/index.html>

⁸ www.microsoft.com/sql/

- Addition of a new RDF model, identified by a name or URI, owned by a user;
- Retrieval of a RDF model by an authorized user;
- Updating/removing of a model by an authorized user;
- Querying of a model with one of the supported languages (RDQL and SPARQL at the moment);
- Setting of access rights to a model by the owner; access rights can be specified on each user or on groups of users; a model can also be registered as publicly accessible, and in this case a URI can be assigned to the model, in order for it to be accessed through an HTTP request to RDFCore, instead of using Java code to access an *RDFEngineInterface* implementation as necessary for other operations.

3.2 Context querying: SPARQL

We identified SPARQL⁹ as the query language that satisfies many of the requirements listed before, since it includes facilities to query more than one RDF model at a time, and the models to use can be specified with URIs. With this approach, a context can be easily represented as an RDF model, identified by a URI – in other words, it can be viewed as a named graph. The only step needed to complete the pipeline and enable a generic repository to answer a SPARQL query on multiple contexts is the retrieval machinery to provide the RDF data for the SPARQL *Dataset* to the SPARQL engine.

In order to do this, it is necessary for the ARQ¹⁰ engine, that we use to actually perform the SPARQL queries, to retrieve the RDF models that build up the *Dataset* for the query without having to know about the specifics of RDFCore.

To do this, we built an implementation of the *Locator* interface, which is an interface to open *InputStreams* from URIs, and together with *FileManager* gives a pluggable architecture to manage retrieval of RDF models stored in non usual ways.

RDFCoreLocator, which is our implementation of *Locator*, uses the HTTP access to RDFCore briefly described before to retrieve models that have been registered as publicly available, and returns

⁹ <http://www.w3.org/TR/rdf-sparql-query/>

¹⁰ <http://jena.sourceforge.net/ARQ/>

an `InputStream` over the resulting HTTP connection. As a result of this design, ARQ is able to obtain the *Dataset* for the SPARQL query in a seamless way.

Actual limitations of this solution are:

- a *RDFCoreLocator* does not know which user is operating the query, so it cannot use the multi user features to access restricted models; at the moment, the only way to overcome this issue is to build a new *RDFCoreLocator* for each different user making a query; at the moment, accessible to anyone;
- the SPARQL protocol,¹¹ at the time of writing, does not consider additional data such as username and password to be specified in the call, so the multiple Locators solution in the previous point cannot be implemented in a straightforward way (e.g., a step of initialization of the query engine is necessary before a SPARQL query can be issued with the SPARQL protocol, but then concurrency issues over the SPARQL server have to be taken into account). As a result, at the moment we don't have a good design solution to enable the SPARQL protocol over restricted access RDF models. While, as described in the specs, conformance with SPARQL protocol does not require that the server accepts any kind of query, this would be a useful feature both in the VIKEF scenario and more in general from an application perspective, since complete access to every model, even if only for reading, would greatly simplify the design of VIKEF and non-VIKEF applications, which would be able to connect to RDFCore without having to access its SOAP services.
- Actual SPARQL protocol conformance of RDFCore is under testing, in order to verify that it respects the W3C specifications.¹²

3.3 System Architecture

As sketched in Figure 2, the main parts of the architecture are:

- A *URI Registry*, used by applications to get the list of context URIs contained in a particular instance of RDFCore.

¹¹ <http://www.w3.org/TR/rdf-sparql-protocol/>

¹² <http://www.w3.org/TR/rdf-sparql-protocol/##conformance>

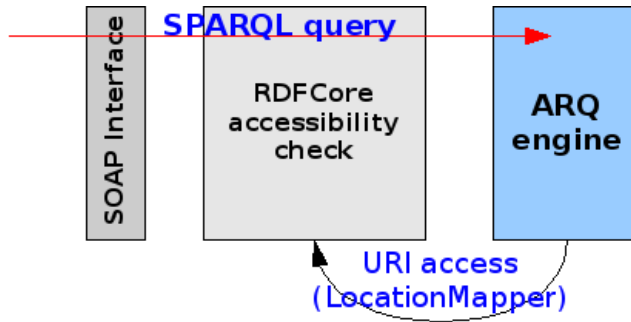


Fig. 1. SPARQL query processing

- A *Compatibility Relations model*, containing statements of the kind $\langle c' R c \rangle$, meaning that context c' is in relation R with context c (all three should be read as URIs for the contexts and the relation).
- A *Compatibility Relation Registry*, where each URI that identifies a CR is related to an implementation for that CR (*semantic attachment*).

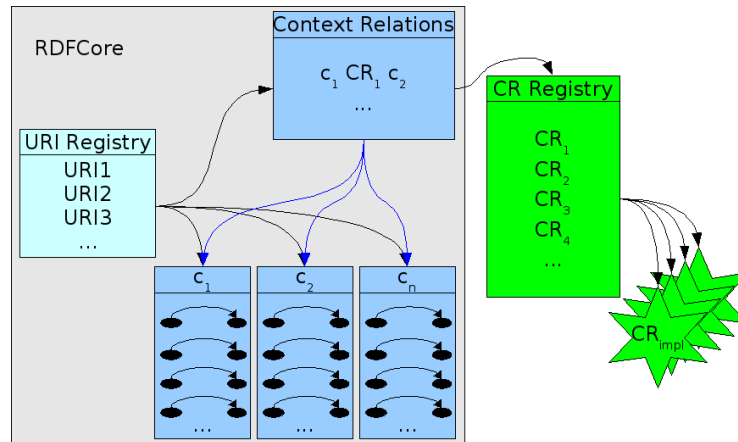


Fig. 2. Compatibility Relation Association Architecture

The architecture presented so far is quite straightforward; however, the reasoning task on the *Compatibility Relations model* (i.e. the model containing the CR statements between contexts) cannot be carried out by a DL reasoner, since the complete semantics of the CRs we want to represent exceed OWL expressiveness. In order to overcome this limitation of the architecture, we devised a plugin-oriented solution, where the URI of a CR identifies a plugin that implements the correct behavior to be carried out. As an example, consider a CR saying that:

“context *context : x* and context *context : y* are *context : compatible* if they have no contradictory statements”¹³

The relation named *context : compatible*, then, has to be inferred (or verified) through the use of some code that has to be associated with the relation, which in this case would do the job of taking the RDF models for the two contexts (i.e. the RDF models labeled *context : x* and *context : y*, available in RDFCore) and use a reasoner on the whole resulting RDF graph in order to evaluate consistency.

4 Conclusions and Future Work

We presented a possible solution to the issues related to uncontextual knowledge, mostly arising from the notion of *universal truth* that RDF model semantics follow, and showed the architecture of our implementation for this solution. Future work we plan to do on this implementation:

- Thorough stress test for the RDFCore component that acts like a “context” server in our architecture, to check for scalability issues wrt number of context, context size, CR number and complexity.
- Implementation of a significant number of CR “attachments”, in order to provide the system with the needed expressive power to match VIKEF requirements.
- Compliance with SPARQL protocol in order to simplify interaction with the knowledge base from the point of view of a VIKEF application.

¹³ Note that no specific reasoner level is set here: a real rule should also specify *how* to verify contradiction

Possible applications for this kind of KR are manifold, as partly described in [1, 2, 7, 8]. Aspects such as beliefs, trust, incomplete knowledge and KB evolution in our opinion can all be tackled with a sensible context system as a base. We believe that in the long run, the vast amount of knowledge represented in the Semantic Web can only be handled properly if represented *in context*.

Additionally, we envision the outcomes of this work to go beyond local aspects and also become relevant from a distributed point of view. As the nature of the Semantic Web is inherently distributed, we think we can contribute to the semantic coordination of Semantic Web agents, firstly by offering the capabilities to make explicit that two knowledge bases belong to their respective agents and to enable the agents to establish semantic links to the KBs of other peers with the help of compatibility relations.

5 Acknowledgments

This research was partially funded by the European Commission under the 6th Framework Programme IST Integrated Project VIKEF - Virtual Information and Knowledge Environment Framework (Contract no. 507173, Priority 2.3.1.7 Semantic-based Knowledge Systems; more information at <http://www.vikef.net>).

References

1. Paolo Bouquet, Luciano Serafini, and Heiko Stoermer. Introducing Context into RDF Knowledge Bases. In *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005. CEUR Workshop Proceedings, ISSN 1613-0073, online <http://ceur-ws.org/Vol-166/70.pdf>*, 2005.
2. Jeremy Carroll, Christian Bizer, Patrick Hayes, and Patrick Stickler. Named Graphs, Provenance and Trust. In *Proceedings of the Fourteenth International World Wide Web Conference (WWW2005), Chiba, Japan*, volume 14, pages 613–622, May 2005.
3. G. Criscuolo, F. Giunchiglia, and L. Serafini. A Foundation for Metareasoning, Part I: The proof theory. *JLC*, 12(1):167–208, 2002.
4. G. Criscuolo, F. Giunchiglia, and L. Serafini. A Foundation for Metareasoning, Part II: The model theory. *JLC*, 12(3):345–370, 2002.
5. F. Esposito, L. Iannone, I. Palmisano, and G. Semeraro. RDF Core: a Component for Effective Management of RDF Models. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003*, 2003.

6. Chiara Ghidini and Fausto Giunchiglia. Local models semantics, or contextual reasoning=locality+compatibility. *Artif. Intell.*, 127(2):221–259, 2001.
7. Ramanathan V. Guha, Rob McCool, and Richard Fikes. Contexts for the semantic web. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2004.
8. Graham Klyne. *Contexts for RDF Information Modelling*. Content Technologies Ltd, October 2000. <http://www.ninebynine.org/RDFNotes/RDFContexts.html>.
9. R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13(1):133–176, 1980.